Rapport Projet de Fin d'Etude: Prostate Cancer Detection in Histology Images

Emile Vaysse

ENSEEIHT & INSA Toulouse - 6ModIA
2024-2025

Contents

1 Introduction

Le cancer de la prostate est l'une des formes les plus courantes de cancer chez les hommes, représentant une cause majeure de morbidité et de mortalité à l'échelle mondiale. Le diagnostic précoce est essentiel pour améliorer les chances de traitement efficace et de survie, mais il reste un défi en raison de la variabilité des formes de ce cancer et de la complexité des données histopathologiques. La biopsie de la prostate, suivie d'une analyse de lames histologiques, est actuellement la méthode de référence pour confirmer la présence de cellules cancéreuses. Cependant, ce processus repose largement sur l'expertise des pathologistes, ce qui peut conduire à des erreurs d'interprétation en raison de la subjectivité humaine et de la charge de travail accrue.

Ces dernières années, avec les progrès rapides de l'intelligence artificielle (IA) et du deep learning, l'automatisation du diagnostic à partir d'images histologiques est devenue un domaine de recherche en plein essor. En particulier, les Vision Transformers (ViT) qui ont montré leur efficacité dans l'analyse d'images médicales complexes. Ces approches permettent non seulement d'améliorer la précision du diagnostic, mais aussi de réduire la variabilité entre plusieurs experts et d'accélérer le processus de diagnostic. Néanmoins, la difficulté pour entraîner des ViTs (comme le coût de calcul) peut être élevée, nous verrons donc également dans ce rapport une autre alternative aux ViTs, toute aussi performante si ce n'est plus, tout en s'affranchissant d'une contrainte dans le milieu médical: la labellisation des données.

Le projet de fin d'études présenté ici s'inscrit dans cette dynamique en proposant une approche de détection du cancer de la prostate à partir d'images histologiques, et en utilisant des techniques avancées de deep learning. L'objectif principal est de développer un modèle capable de détecter et de classifier automatiquement les tissus cancéreux dans les images d'histologie prostatiques. Cette automatisation pourrait permettre aux pathologistes de se concentrer sur les cas complexes tout en assurant une analyse plus rapide et précise des échantillons. Le partenaire de ce projet est un hôpital, pour lequel avec mon équipe, nous avons proposé un réseau pré-entraîné sur le super-calculateur du Barcelona Super-computing Center. Ce réseau pré-entraîné a par la suite été utilisé pour classifier des images d'histologie issues d'un dataset privé propre au partenaire.

Ce travail a été effectué au sein du BSC (Barcelona Supercomputing Center), l'un des centres de recherche les plus importants d'Espagne. Ce centre est présent dans de nombreux sujets de recherche à savoir les sciences de la vie et de la Terre, l'HPC (High Performance Computing), Computer Sciences, et également l'Intelligence Artificielle notamment au sein du département HPAI (High Performance and Artifical Intelligence) au sein duquel j'ai effectué mon stage.

Pour réaliser ce projet, plusieurs étapes sont nécessaires, allant de la préparation et de l'annotation des données à l'entraînement des modèles de classification. Les images histologiques de la prostate présentent des caractéristiques variées, telles que des noyaux cellulaires denses et des structures glandulaires altérées, rendant leur analyse particulièrement difficile. La gestion de ces variations constitue un défi majeur pour l'entraînement des modèles. Des techniques telles que la segmentation d'images, l'augmentation des données et l'apprentissage supervisé ou semi-supervisé peuvent être utilisés pour répondre à ces défis.

Ce rapport présente les fondements théoriques de l'approche choisie, les étapes du développement du modèle, ainsi que les résultats obtenus lors des phases d'entraînement et de test. L'objectif final est de montrer que l'IA, appliquée aux images histologiques prostatiques, peut contribuer à améliorer la détection précoce du cancer de la prostate.

Ainsi, nous commencerons ce rapport par présenter les différents datasets publics d'images d'histologie utilisés pour le pré-entraînement de notre modèle, ensuite, nous parlerons d'un problème fondamental induit par le receuil des différents datasets médicaux publics utilisés dans cette étude, et comment répondre à ce problème (apprentissage auto-supervisé). Par la suite, nous expliquerons quelle méthode d'entraînement auto-supervisé nous avons choisi et pourquoi, ainsi que son fonctionnement. Nous étudierons ensuite les différentes options possibles de modèles de vision par ordinateur et choisirons parmi les différentes possibilités. L'entraînement et les résultats de notre modèle ainsi que son déroulement seront également détaillés. Enfin, nous conclurons ce rapport par une mise en perspective de ce projet de fin d'étude.

2 Datasets et Baselines choisis

Pour tenter de répondre au besoin de détection automatique de patterns cancéreux dans des images d'histopathologie, il nous faut bien évidemment disposer de banque de données conséquentes. En effet, plusieurs datasets publiques sont disponibles sur internet, néanmoins, la classification ou segmentation de ceux-ci est parfois manquante ou incomplète. Dès lors, il nous a fallu trouver des moyens de compenser ce manque d'information, tant pour l'entraînement que pour l'évaluation de notre modèle.

2.1 PANDA

Le premier dataset que nous avons décidé d'utiliser est le dataset PANDA disponible sur la plateforme Kaggle. La difficulté avec ce dataset, contrairement aux suivants, est qu'il est très volumineux (400 Go), c'est pourquoi je n'ai naturellement pas pu le charger sur mon ordinateur personnel. En effet, il a fallu le charger sur le SuperComputer MareNostrum du BSC. On peut voir un exemple de ces images sur la figure 1.

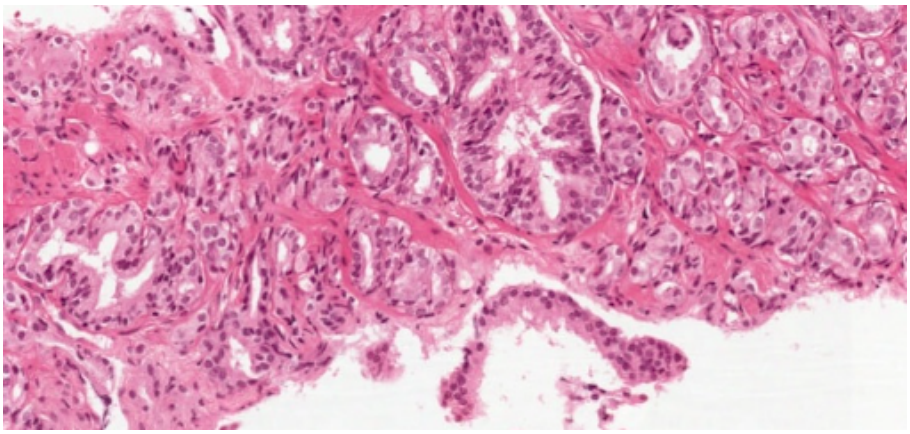

Figure 1: Exemple d'image d'histologie PANDA

Une fois chargé sur le SuperComputer, il a fallu analyser le dataset. PANDA est composé d'images TIFF (.tiff), ces images au format particulier sont spécialement conçues pour l'imagerie médicale et nottament les images d'Histopathologie. Ces images sont généralement très grandes (plusieurs milliers de pixels au carré). De part cette taille importante des images, il nous a fallu nous demander comment analyser ces images au nombre de 10 000 dans le dataset PANDA. En effet, il apparaît difficile de réussir à entraîner un modèle sur seulement 10 000 images de très grandes taille. Néanmoins, dans le dataset PANDA, une segmentation de chaque image Tiff est disponible. Ceci m'a permis de segmenter chaque image et ainsi de pouvoir y associer une classification. Pour le dataset PANDA, 5 classes étaient enregistrées: 0: background de l'image, 1: Tissu sain, 2: Gleason 3, 3: Gleason 4, 4: Gleason 5. Gleason 3 à 5 représente la sévérité du cancer sur l'échelle Gleason.

En accord avec le partenaire médical du projet (qui avait déjà fait le choix de la dimension des images traitées), nous avons découpé chaque image Tiff en sous-images de 224x224 sans overlap. La politique choisie était donc de donner à chaque segmentation d'image (224x224) la classe la plus élevée dans l'échelle Gleason présente dans la segmentation de l'image associée. Ainsi nous avons pu obtenir un dataset composé d'images en 224x224 avec un label associé.

Néanmoins, la classification associée à chaque image ne servira pas pour l'entraînement. En effet le projet avait pour but d'entraîner un modèle sans labellisation (auto-supervisé) mais cette classification servira tout de même lors de la phase d'évaluation des résultats sur différents benchmarks.

2.2 DeepGleason

Nous avons également utilisé le dataset publique DeepGleason. Tout comme PANDA, DeepGleason contient des images Tiff et des images représentant la segmentation de ces images. Voici sur la figure 2 un exemple d'image DeepGleason.

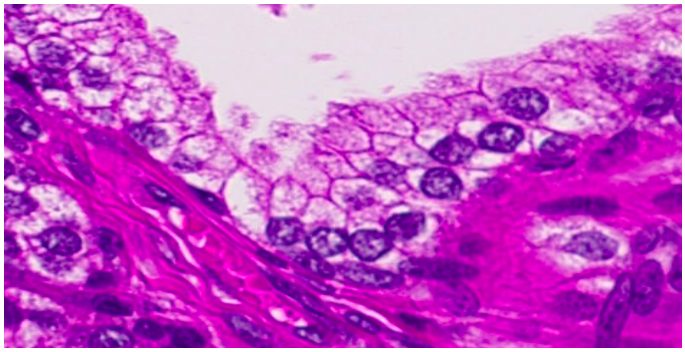

Figure 2: Exemple d'image d'histologie DeepGleason

L'échelle utilisée pour la segmentation des images est toujours l'échelle Gleason. Le dataset DeepGleason, tout comme le dataset PANDA, contient des images d'Histopathologie de la prostate. Néanmoins, ce dataset a dans un premier temps été utilisé seulement pour tester le modèle entraîné sur PANDA, notamment en raison de sa taille relativement petite (approximativement 2 Go). Pour ce dataset, nous avons choisi de découper les images Tiff en 224x224 tout comme pour PANDA et de faire de même pour les masques de segmentation.

Ensuite, au lieu d'associer un label à chaque image pour l'évaluation finale, nous avons seulement stocké le découpage du masque et tenté de retrouver ce masque de segmentation à l'aide d'un décodeur convolutionnel pour s'assurer que notre modèle encodeur (le modèle XCiT entraîné sur PANDA (nous en reparlerons plus tard)) ait "bien compris" la distribution associée à chaque image d'histopathologie.

2.3 SICAPv2

Nous avons également utilisé le dataset SICAPv2 comme benchmark. Ce dataset contient également des images Tiff que nous segmentons de la même manière que nos deux datasets précédents. A première vue ce dataset semble contenir une classification de chaque sous-image Tiff de dimension 512x512. Ces images sont au nombre de 1554. Voici un exemple d'image de ce dataset dans la figure 3.

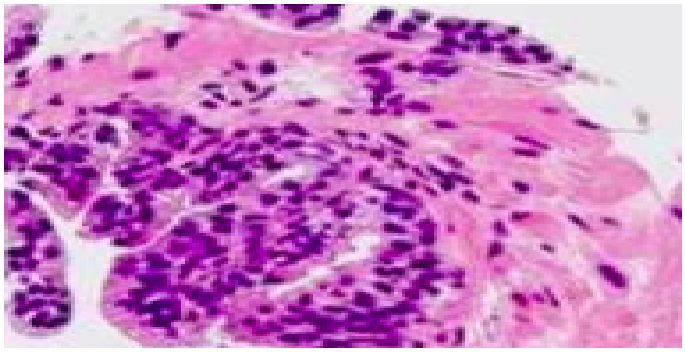

Figure 3: Exemple d'image d'histologie SICAPv2

Néanmoins, après étude du dataset, nous nous sommes rendus compte que chaque sous image Tiff de 512x512 était annotée de la même manière que toutes celles de la même image Tiff originelle

(image de très grande dimension). Ceci pose alors un problème étant donné que toute sous image Tiff ne présente pas les caractéristiques associée à la classification de l'image originelle.

On ne peut ainsi pas réellement se fier à cette classification, c'est pourquoi il a fallu imaginer une autre manière de tester ce benchmark. En effet, il est à noter que nous ne disposions pas non plus de masques de segmentation fiables. Nous avons alors opté pour un décodeur convolutionnel (comme pour DeepGleason), mais cette fois l'image cible n'est pas un masque de segmentation mais la même image que l'image d'entrée mais "blurrée" (en effet, les images d'histopathologie ne sont pas des signaux très "continus").

Toute cette étape d'évaluation avait pour but de s'assurer que le modèle arrivait à capter les informations de l'image de départ.

2.4 Baselines

Nous présentons ici les modèles baselines qui nous serviront de comparatif avec le modèle que nous entraînerons.

2.4.1 Hibou

Le modèle **Hibou** est un modèle basé sur un Transformer conçu pour l'analyse d'images d'Histopathologie. Hibou est entraîné sur des données histopathologiques provenant de divers organes, ce qui lui confère une capacité de généralisation robuste, même sur des benchmarks spécifiques tels que PANDA, DeepGleason ou SICAPv2. Ce modèle a été entraîné sur près de 1.2 milliard d'images d'Histopathologie.

La structure de Hibou repose sur un encodeur Transformer qui traite les images en extrayant des *patch embeddings*. Chaque image est divisée en patches de taille fixe (par exemple, 16×16), et chaque patch est linéarisé avant d'être transformé en un embedding de 768 dimensions. Ces embeddings sont ensuite passés à travers des blocs Transformer comportant des mécanismes d'*attention multi-tête* et des couches de normalisation (LayerNorm).

2.4.2 Histoencoder

Le modèle **Histoencoder** est un modèle *baseline* proposé pour l'analyse d'images d'histopathologie. Il repose sur une architecture d'encodeur XCiT. Le principal objectif de Histoencoder est d'extraire des *features* visuelles pertinentes notamment en tirant partie des mécanismes d'attention appliqués aux images. Ce modèle a été entraîné sur 48 millions d'images.

3 Secteur Médical: Problèmes de labellisation des données

Dans le secteur médical, la labellisation des données est un processus crucial pour la mise en place et l'optimisation des systèmes d'intelligence artificielle (IA), notamment pour les applications en diagnostic, en traitement personnalisé, et en gestion de la santé publique. Cependant, ce processus complexe présente de nombreux défis qui peuvent affecter directement la performance et la fiabilité des modèles basés sur l'apprentissage automatique.

Les données médicales sont souvent de nature hétérogène, comprenant des images médicales (radiographies, IRM, etc.), des dossiers patients, des résultats d'analyses biologiques, et des notes cliniques. Ces données proviennent de multiples sources, rendant difficile leur standardisation. Les images peuvent avoir des résolutions et des formats différents, tandis que les dossiers peuvent être remplis de manière variable selon les professionnels de santé. Cette diversité rend le processus de labellisation particulièrement compliqué, car chaque type de donnée peut nécessiter des experts différents pour l'annotation, comme des radiologues pour les images médicales ou des cliniciens pour les dossiers patients.

L'un des problèmes majeurs est le manque d'experts disponibles pour effectuer la labellisation. En raison de la spécificité des données médicales, seuls des professionnels qualifiés, tels que des médecins spécialistes, peuvent effectuer cette tâche avec précision. La labellisation des images médicales, par exemple, nécessite une expertise en radiologie pour distinguer des anomalies subtiles qui pourraient être ignorées par des non-experts. Le temps et le coût nécessaires pour former ou employer des annotateurs qualifiés sont souvent trop importants, surtout à grande échelle.

Même lorsque des experts sont disponibles, il peut y avoir une variabilité significative dans leurs annotations. Des études ont montré que différents médecins peuvent interpréter les mêmes résultats d'IRM ou de radiographie de manière légèrement différente, en fonction de leur expérience et de leur spécialisation comme dans Elmore et al. [1]. Cette variabilité introduit une incertitude dans les données annotées, ce qui peut nuire à la performance des modèles d'apprentissage automatique. Pour minimiser cette variabilité, il est parfois nécessaire de combiner les annotations de plusieurs experts, augmentant encore le coût et la complexité du processus.

La labellisation des données médicales soulève également des préoccupations éthiques, notamment en matière de confidentialité des patients. Les données médicales sont extrêmement sensibles et leur anonymisation doit être assurée avant toute utilisation pour l'entraînement de modèles. Toutefois, certaines informations essentielles à la labellisation, comme l'historique médical d'un patient ou des détails contextuels, peuvent être difficiles à anonymiser. La gestion de ces données selon les régulations strictes, comme le Règlement Général sur la Protection des Données (RGPD) en Europe, complexifie encore le processus de labellisation.

Un autre défi fréquent dans le secteur médical est l'imbalance des classes dans les données. Par exemple, dans les bases de données d'images médicales, les cas de pathologies rares peuvent être sous-représentés, tandis que les cas normaux sont majoritaires. Les modèles d'apprentissage automatique risquent alors de surperformer pour les classes dominantes, mais de mal détecter les anomalies rares. De plus, si les données médicales utilisées pour l'entraînement proviennent d'un groupe de population spécifique, comme une population majoritairement caucasienne, cela peut introduire des biais qui compromettent l'efficacité des modèles sur des groupes sous-représentés.

Face à ces défis, plusieurs solutions peuvent être imaginées. Des techniques comme l'apprentissage semi-supervisé peuvent réduire la quantité de données labellisées nécessaires en permettant aux modèles d'apprendre à partir de données non annotées. De plus, des approches collaboratives entre institutions médicales pour partager des ensembles de données anonymisées et labellisées contribuent à améliorer la qualité et la diversité des données utilisées. Enfin, l'utilisation de technologies comme le transfert d'apprentissage permet d'appliquer des modèles pré-entraînés sur d'autres jeux de données similaires, réduisant ainsi le besoin d'un grand volume de données annotées spécifiques. Enfin, l'apprentissage auto-supervisé, qui ne nécessite aucun labels, peut résoudre ces problèmes de labellisation.

4 Apprentissage auto-supervisé: DINO

Nous allons ici commencer par une brève introduction sur ce qu'est l'apprentissage auto-supervisé et pourquoi il peut s'avérer être si important dans l'imagerie médicale. Nous décrirons ensuite un framework d'apprentissage auto-supervisé: DINO. C'est ce framework que nous avons utilisé pour l'entraînement de notre modèle de détection de cancers.

4.1 Qu'est ce que l'Apprentissage Auto-Supervisé ?

L'apprentissage auto-supervisé est une méthode d'entraînement des modèles d'intelligence artificielle où le système apprend à partir de données non étiquetées en générant lui-même des pseudo-étiquettes ou des tâches auxiliaires. Cette approche a émergé comme un domaine clé dans l'apprentissage profond, en particulier pour les données massives non annotées, comme des images, du texte ou des signaux audio, où l'étiquetage manuel est coûteux et souvent impraticable. Comme nous l'avons vu précédemment, le domaine médical et notamment l'imagerie médicale, est un secteur particulièrement touché par la difficulté d'étiquetage, ce qui rend l'apprentissage auto-supervisé particulièrement attrayant dans notre étude.

L'idée centrale de l'apprentissage auto-supervisé est de contourner la phase de labellisation en définissant une fonction de coût qui repose sur une autre manière de définir le signal d'apprentissage, ce qui permet au modèle de découvrir des structures sous-jacentes dans les données. Ces nouvelles fonctions de coût exploitent des informations intrinsèques des données pour fournir des signaux d'apprentissage. Par exemple, pour les images, une tâche courante consiste à prédire des transformations appliquées à l'image, comme la rotation (Doersch et al.) [2], ou à reconstruire des parties manquantes d'une image. Pour les représentations textuelles, des méthodes comme BERT (Devlin et al.) [3] se basent sur des tâches de masquage de mots pour apprendre des représentations riches des données textuelles.

Un tournant majeur de l'apprentissage auto-supervisé dans le domaine de la vision par ordinateur a été marqué par des travaux comme SimCLR (Chen et al.) [4] et MoCo (He et al.) [5]. Ces modèles ont utilisé des approches de contraste pour apprendre des représentations qui maximisent la similarité entre des vues augmentées d'une même image et minimisent celle entre des images différentes. Ces méthodes ont démontré que les représentations apprises en auto-supervisé peuvent rivaliser avec celles obtenues par des méthodes supervisées sur de grandes bases de données étiquetées. Plus récemment, le framework DINO (Caron et al.) [7], réalisé par des chercheurs de Meta a permis de pousser l'étude des frameworks "student - teacher" à un niveau où même la pseudo-labellisation n'est pas nécessaire (en effet certains frameworks "student - teacher" utilisaient le modèle teacher pour labelliser certaines données et ensuite ré-entraîné le modèle avec ces pseudo-labels).

Une idée dans l'apprentissage auto-supervisé a également été très utilisée, et c'est sur cette idée que repose le framework DINO. Cette idée est la structure "student - teacher", ou "distillation". En effet, dans cette idée, on va avoir un modèle student qui va apprendre d'un modèle teacher. Ce modèle teacher peut par exemple être un modèle pré-entraîné sur ImageNet, qui va au cours de l'entraînement donner (ou distiller) du savoir au modèle student. Dans notre cas avec le framework DINO, l'interaction entre ces deux modèles student et teacher est un peu plus complexe que cela mais nous allons tenter dans les prochaines parties d'expliquer comment fonctionne ce framework. On résume ci-dessous l'idée d'architecture "Student - Teacher" à l'aide d'une figure (4):

L'apprentissage auto-supervisé devrait donc avoir dans le futur des répercussions très positives dans le domaine de l'imagerie médicale de part sa nature. En réduisant la dépendance aux annotations manuelles, l'apprentissage auto-supervisé ouvre la voie à des applications dans le domaine de l'imagerie médicale, et c'est dans ce contexte que cette étude sur l'imagerie médicale en Histopathologie a été mené.

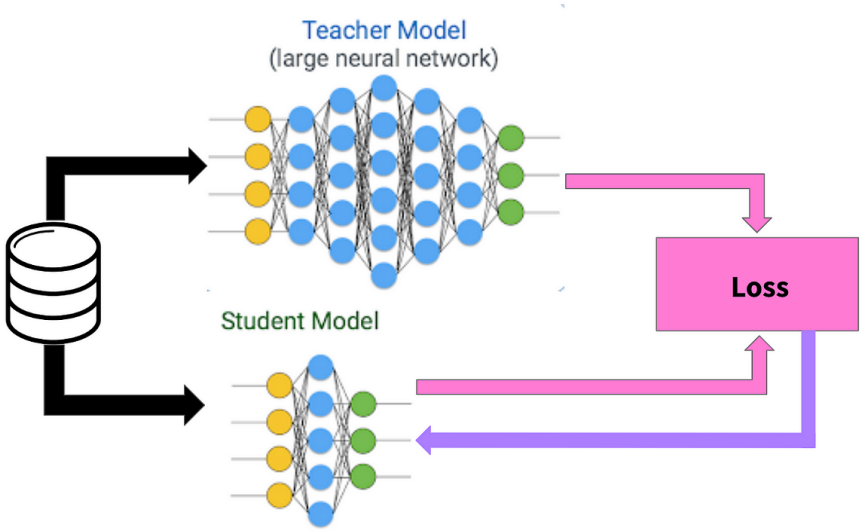

Figure 4: Architecture de Distillation (Student - Teacher)

4.2 Introduction à DINO

Dans cette partie nous allons introduire le framework DINO (Touvron et al. 2021) [7]. DINO est un framework capable de conduire l'entraînement d'un modèle de computer vision sans aucune supervision. Il s'agit donc d'une méthode d'apprentissage auto-supervisé. DINO repose sur un concept que l'on a déjà introduit plus tôt à savoir sur la distillation de connaissances (Knowledge distillation). En effet, DINO utilise la structure "teacher - student" pour entraîner un modèle teacher qui sera capable (en théorie) de comprendre la structure de l'image, ses différents composants et ainsi fournir un embedding pouvant être utilisé pour une tâche (comme de la classification ou de la segmentation).

4.2.1 Structure Student - Teacher

Une fois cela dit, en quoi consiste ce framework exactement ? DINO est comme dit précédemment un framework de distillation de connaissances, néanmoins, contrairement à certains modèles de ce type, DINO ne requiert pas un modèle teacher pré-entraîné. En effet, les deux modèles teacher et student ne sont pas pré-entraînés et possèdent exactement la même architecture. Mais alors comment la distillation de connaissance est elle possible ? Cette question est importante et nous tenterons d'y répondre ultérieurement mais avant de comprendre comment cela peut il marcher, il nous faut d'abord comprendre la structure générale du framework. Comme dit précédemment, nous avons au départ deux modèles, un student et un teacher. Ces deux modèles peuvent être n'importe quel modèle, dans l'article fondateur de DINO (Touvron et al. 2021) [7] les modèles testés sont des CNNs, des ViTs et des XCiTs. La loss utilisée pour ce framework est:

$$min_{\theta_s} H(P_t(x), P_s(x))$$

Avec θ_s les paramètres associés au modèle student, avec $H(a,b) = -a.log(b)$, et P_* la sortie de $*$ (s pour student et t pour teacher) normalisée $(P_*(x)^{(i)} = \frac{g_*(x)^{(i)}/\tau}{\sum_{k=0}^{k=N} g_*(x)^{(k)}}$.

4.2.2 Collapse

On remarque ici que l'on divise la sortie du modèle g_* par un paramètre τ. On appelle ce paramètre la "température". Ce paramètre permet d'accentuer la sortie normalisée du modèle, en effet, un τ faible permettra d'accentuer les écarts entre les différents éléments de la distribution normalisée

(cette distribution est obtenue avec P et c'est en fait une $Softmax$ sur la sortie du modèle (teacher ou student)). Cette utilisation de τ conduit donc à "sharpen" la distribution en sortie de la $Softmax$ du modèle. Néanmoins, une telle opération peut conduire à un "collapse" (explosion des valeurs) de la sortie de la $Softmax$. Une normalisation pourrait être utilisée mais dans le cas de DINO il a été choisi d'utiliser un centering pour éviter qu'une dimension domine les autres en déterminant une valeure c:

$$c = mc + (1 - m)\frac{1}{B}\sum_{i \in B} g_{\theta_t}(x_i)$$

Avec m un paramètre fixé choisi par l'utilisateur et B le batch courant.

Une fois c obtenu on actualise la sortie du modèle teacher de la façon suivante: $g_{\theta_t}(x) = g_{\theta_t}(x) + c$, et le "sharpenning" se fait au niveau de la $Softmax$.

Cette méthode peut tendre à une uniformisation des valeurs de sortie, mais le "sharpenning" permet de contre-balancer cet effet évitant ainsi le "collapse" des valeurs de sortie.

4.2.3 Entraînement du teacher ?

On remarque ègalement que l'ensemble des paramètres sur lesquels on optimise est θ_s. Cela nous permet de déduire que l'algorithme d'optimisation ne tentera pas d'optimiser les paramètres du modèle teacher. En effet avec DINO on n'optimise pas le teacher, en tout cas, pas avec un algorithme d'optimisation. On peut donc donner cette architecture pour l'algorithme DINO (5):

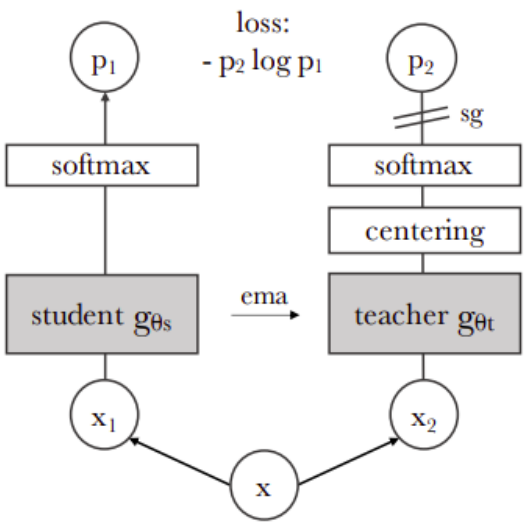

Figure 5: Architecture du framework DINO (les barres "sg" correspondent à une "coupure de gradient", les gradient ne sont donc pas propagés à travers le teacher)

Mais alors comment le modèle teacher s'acualise-t-il ? Le modèle teacher s'actualise en mettant à jour les paramètres du teacher à chaque epoch. Comment ? En faisant simplement une exponential moving average:

$$\theta_t = \lambda\theta_t + (1 - \lambda)\theta_s$$

Avec λ proche de 1 (dans le cas d'utilisation général λ peut être fixé à 0.996).

4.2.4 Relations Globales - Locales

Néanmoins, nous avons ici choisi d'occulter un aspect important du framework DINO. En effet, la fonction de coût que nous avons définie plus haut n'est pas exactement celle utilisée en réalité.

Pour comprendre pourquoi cette fonction de coût annoncée n'est pas idéale (ceci a été vérifié, en effet, suite à une erreur d'implémentation de ma part j'ai utilisé cette fonction de coût pour entraîner un modèle ViT et les résultats étaient mauvais avec une fonction de coût qui ne diminuait pas) il faut identifier un défi très connu en computer vision: établir des modèles capables de faire correspondre les patterns locaux aux patterns globaux au sein d'une image.

Une fois cela dit, comment dans notre cas réussir à mêler patterns locaux et globaux ? Dans l'article de DINO, une solution ingénieuse est proposée en établissant cette fonction de coût:

$$L(P_t(x), P_s(x)) = \sum_{x \in V} \sum_{x' \in V'} H(P_t(x), P_s(x'))$$

Avec V des crops "globaux" de l'image (dans le papier ils est proposé des crops globaux de 224x224) et V' des crops "locaux" (dans le papier ils est proposé des crops locaux de 96x96). Il est également à noter que ces "crops" sont obtenus de manière aléatoire pour généraliser un maximum l'entraînement.

On résume cette idée ci-dessous avec cette figure (6):

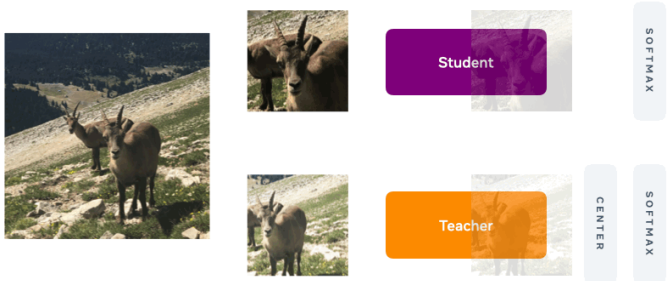

Figure 6: Illustration de la décomposition en "crops"

4.2.5 Difficultés rencontrées en raison des relations Globales - Locales

On remarquera avec cette nouvelle fonction de coût mêlant patterns globaux et patterns locaux que l'on démultiplie les éléments de la fonction de coût en fonction du nombre de vues globales et locales que l'on sélectionne. Cette particularité a en effet causé certains problèmes de gestion mémoire des GPUs puisque l'on doit stocker les gradients de chaque composante de la fonction de coût sur nos différents GPUs. Néanmoins cette décomposition des relations Globales - Locales des différentes images reste nécessaire pour la convergence de notre problème d'optimisation, il a donc fallu trouver des solutions à ce problème.

L'une des solutions que nous avons identifiée a été de tout simplement réduire le nombre de vues locales (en effet nos images, contrairement à ImageNet utilisé par Touvron et al. [7], ne sont pas de dimensions très importantes). Mais une autre approche est possible à savoir l'accumulation de gradient. Cette technique repose sur un concept simple, au lieu de faire itérer notre algorithme d'optimisation à chaque batch, on "accumule les gradients" pendant par exemple quatre batchs, puis on optimise sur les gradients "accumulés". Ainsi, on peut accélérer l'entraînement sans perdre de manière importante en efficacité d'entraînement.

4.2.6 Résultats de l'algorithme DINO

Pour finir, DINO a prouvé son efficacité nottament lorsque le modèle backbone utilisé est un ViT. En effet ses résultats dans cette configuration sont impressionnants, nottament lorsque l'on observe les "attention maps" des différentes têtes d'attention du modèle ViT teacher . En effet la segmentation obtenue est impressionnate, la voici 7:

Pour terminer, on fournit un pseudo-code ci-dessous de l'algorithme DINO 8:

Figure 7: Vsisualisation de différentes têtes d'attention

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```

Figure 8: Pseudo code DINO

5 Modèle pour DINO: ViT ou XCiT

5.1 ViT

Le partenaire médical avec lequel j'ai travaillé a proposé pour le projet d'utiliser le modèle XCiT de Meta (qui sera expliqué plus tard). Néanmoins pour bien comprendre ce qu'est le modèle XCiT il faut comprendre ce qu'est un ViT (Vision Transformer) et c'est ce que nous tenterons d'expliquer dans cette partie.

5.1.1 Structure d'un ViT et mécanisme d'attention

Un Vision Transformer (ViT) est un modèle d'apprentissage profond conçu pour des tâches de vision par ordinateur. Il s'inspire des Transformers utilisés en traitement automatique du langage, mais adapte leur architecture pour traiter des images. Contrairement aux modèles convolutionnels traditionnels (CNN), les ViT utilisent des mécanismes d'attention globale pour capturer les dépendances entre les différentes régions d'une image, sans exploiter explicitement les relations locales comme les convolutions.

Pour traiter une image, celle-ci est divisée en petits patches réguliers de taille fixe, chacun étant aplati et projeté dans un espace de dimension d via une transformation linéaire. Soit une image I de dimensions $H \times W \times C$, où H, W, et C représentent respectivement la hauteur, la largeur, et le nombre de canaux (par exemple, $C = 3$ pour des images en couleur). Après division en patches, chaque patch est de taille $P \times P$, ce qui donne $N = \frac{HW}{P^2}$ patches. Chaque patch est vectorisé en $x_i \in \mathbb{R}^{P^2C}$ et projeté dans un espace de dimension d via

$$z_i^0 = Wx_i + b,$$

où $W \in \mathbb{R}^{d \times P^2C}$.

Un vecteur d'emplacement, appelé *embedding* de position, est ajouté pour introduire l'information spatiale, donné par

$$z_i^0 \leftarrow z_i^0 + e_i,$$

où $e_i \in \mathbb{R}^d$. Ces vecteurs $z^0 \in \mathbb{R}^{N \times d}$ sont ensuite fournis au Transformer.

Le Transformer repose sur des couches de mécanismes d'attention multi-têtes et des réseaux feedforward. L'attention multi-têtes calcule, pour chaque vecteur z_i, une combinaison pondérée des autres vecteurs via le produit scalaire entre les représentations projetées des vecteurs. Les projections sont définies par :

$$Q = zW_Q, \quad K = zW_K, \quad V = zW_V,$$

où $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$. L'attention est calculée comme :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V.$$

L'attention multi-têtes est une concaténation de plusieurs calculs d'attention pondérés, suivie d'une projection linéaire.

Ainsi, l'attention est un mécanisme particulier qui cherche à faire correspondre plusieurs parties de l'image, et de modéliser ceci par une "Attention Map", que l'on appelle ici "Attention". Ce mécanisme est également le plus répendu dans le domaine du NLP (Natural Language Processing) et il se trouve qu'il égale désormais les performances des CNNs. Il est également à noter que les ViTs contiennent de la self-attention et de la cross attention. La self-attention fera correspondre des requêtes et des clés avec des valeurs issues des mêmes inputs, tandis que la cross-attention fera correspondre des clés et des requêtes venant de l'encodeur avec des valeurs de l'input dans l'espace des embeddings.

Le Transformer inclut des connexions résiduelles et une normalisation de couche pour stabiliser l'entraînement et permettre un meilleur flux d'informations. Après l'attention multi-têtes, une

couche feedforward appliquée indépendamment sur chaque vecteur utilise deux transformations linéaires séparées par une activation non linéaire :

$$\text{FFN}(z) = \sigma(zW_1 + b_1)W_2 + b_2,$$

où $W_1 \in \mathbb{R}^{d \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d}$, et σ est une fonction d'activation (souvent GELU).

Le processus est itéré sur plusieurs couches pour produire des représentations z^L finales. Dans le ViT, un vecteur spécial, appelé *token de classe* (z_{class}), est initialisé et passe à travers les couches du Transformer. À la fin, z_{class} est utilisé pour des prédictions via une tête dense. Le modèle est entraîné par descente de gradient pour minimiser une fonction de perte, typiquement l'entropie croisée pour la classification.

5.1.2 Architecture

On montre ici l'architecture d'un ViT de manière plus imagée 9:

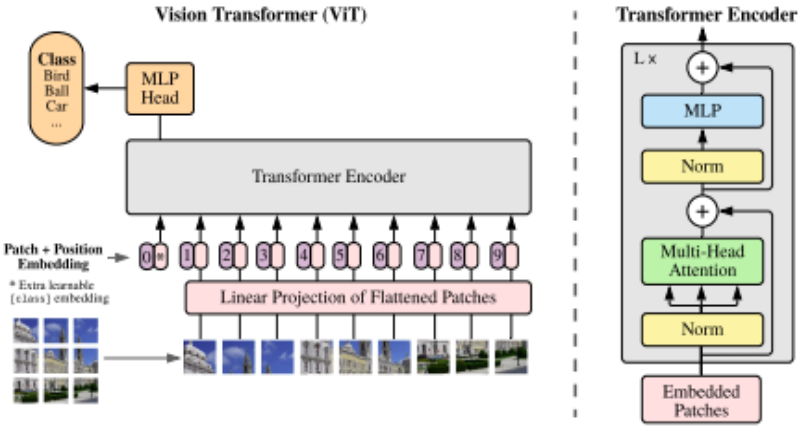

Figure 9: Architecture d'un ViT

Nous avons ici présenté le Vision Transformer introduit dans le papier de Google (Dosovitskiy et al.) [6]. Il est à noter que les Vision Transformer sont désormais une alternative solide aux CNNs bien qu'il requiert plus de données pour l'entraînement. Néanmoins, il a été montré dans le papier de DINO que les ViTs entraînés de manière auto-supervisé étaient capable d'obtenir des segmentations (à l'intérieur des différentes têtes d'attention) bien plus précises que d'autres méthodes supervisées.

5.1.3 Visualisation

Comme introduit précédemment, l'explicabilité du fonctionnement des Vision Transformers peut être visualisée de manière intuitive en visualisant les résultats des différentes têtes d'attention. Grâce à ces têtes d'attention on peut déterminer quelle partie du modèle (ViT) est associée à quelle instance de l'image. Mais comment visualiser avec les valeurs de l'attention map ? En fait, on a à disposition une matrice V de dimension $N \times d$ avec N le nombre de tokens. On va donc calculer la norme de chaque vecteur de V sur la dimension 0:

$$Visualition(x, y) = ||w_{(x,y)}||$$

avec $x \in N_x$, $y \in N_y$ et $N_x \times N_y = N$ et $w_{(x,y)}$ le vecteur dans $Attention(Q, K, V)$ associé au token (x, y).

Le choix du nombre de têtes d'attention est donc important, néanmoins il est difficile d'estimer le nombre de têtes optimales.

On montre ici quelques exemples de visualisation de têtes d'attention pour le dataset ImageNet 10:

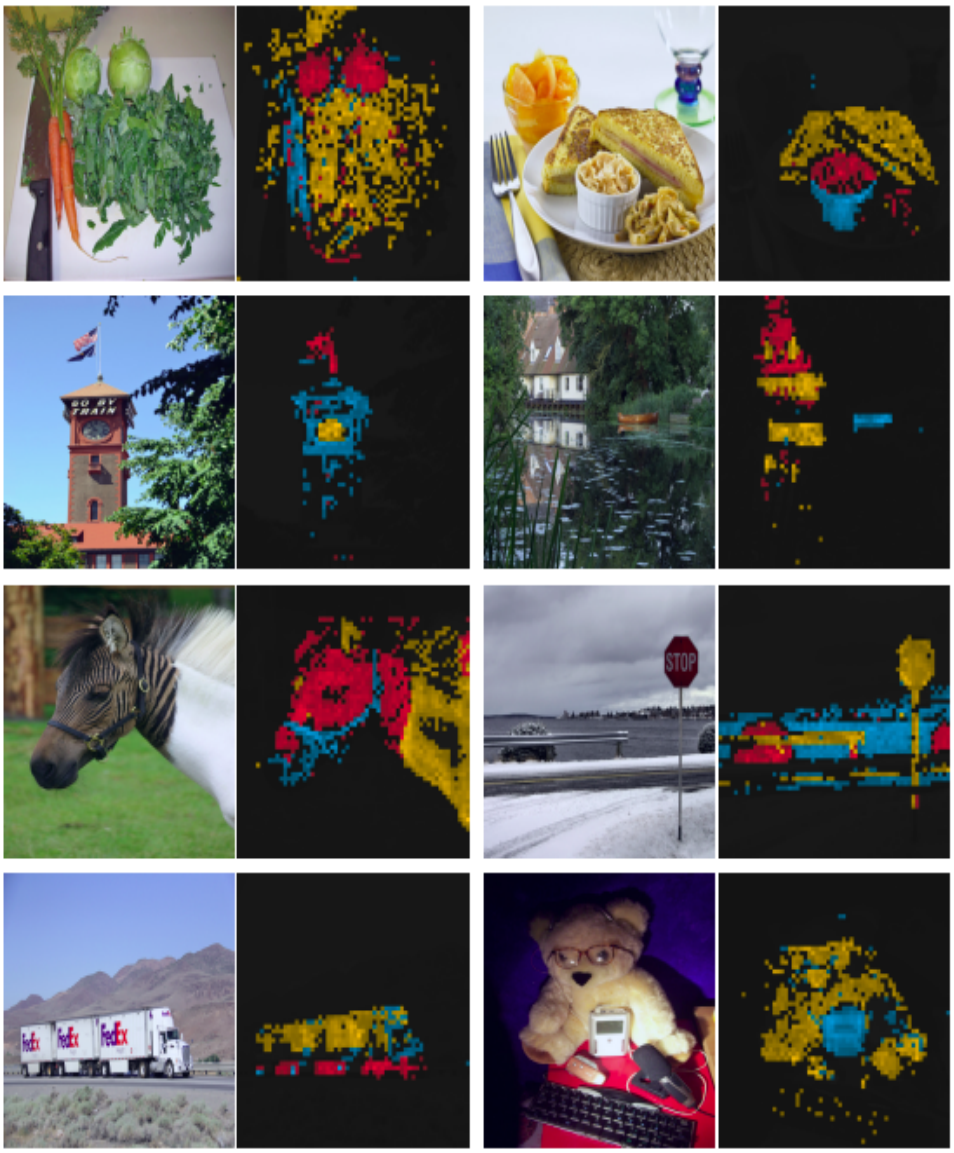

Figure 10: Visualisation de différentes têtes d'attention

5.2 XCiT

Maintenant que l'on a introduit ce qu'était un ViT, nous allons expliquer un nouveau type de modèle qui s'inspire grandement des ViTs mais qui contient certaines différences. Le modèle XCiT est celui sur lequel mon stage s'est principalement déroulé, ce modèle a été proposé par des équipes de Meta (El Nouby et al.) [9]. En effet, ce modèle est celui qui a été demandé par le partenaire médical avec lequel j'ai travaillé. Ce modèle présente plusieurs avantages comparé aux Vision Transformers et nous tenterons de les exposer.

5.2.1 Structure d'un XCiT et mécanisme d'attention modifié

Le modèle XCiT (Cross-Covariance Image Transformer) est une amélioration de l'architecture Vision Transformer (ViT), conçue pour traiter certaines limitations des ViT tout en conservant leurs points forts. Comme les ViT, les XCiT divisent les images en patches, mais modifient le mécanisme d'attention pour améliorer l'efficacité computationnelle et capturer plus efficacement les relations spatiales entre les patches.

Comme pour les ViT, une image I de dimensions $H \times W \times C$ est d'abord divisée en $N = \frac{HW}{P^2}$ patches de taille $P \times P$, chaque patch étant vectorisé en $x_i \in \mathbb{R}^{P^2C}$. Une projection linéaire transforme chaque patch en un espace latent de dimension d :

$$z_i^0 = Wx_i + b, \quad W \in \mathbb{R}^{d \times P^2C}.$$

Un embedding de position $e_i \in \mathbb{R}^d$ est ajouté comme dans les ViT pour conserver l'information spatiale :

$$z_i^0 \leftarrow z_i^0 + e_i.$$

Cependant, contrairement aux ViT, les XCiT utilisent un mécanisme d'attention différent, appelé *Cross-Covariance Attention* (XCA). Plutôt que de calculer une attention basée sur des relations entre vecteurs Q, K, V (comme dans les ViT), XCA se concentre sur les corrélations entre les dimensions des embeddings composant les matrices des requêtes, clées et valeurs, en opérant sur la transposée de l'espace des patches. La matrice des représentations $Z \in \mathbb{R}^{N \times d}$ est transposée en $Z^\top \in \mathbb{R}^{d \times N}$, et l'attention est calculée comme :

$$\text{XCA}(Z) = \text{softmax}\left(\frac{W_Q^\top Z^\top Z W_K}{\tau}\right) \cdot ZW_V,$$

où $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$ sont des matrices d'apprentissage. Cette approche réduit la complexité liée au nombre de patches (N) en se concentrant sur les dimensions latentes (d), ce qui est particulièrement utile pour de grandes images. En effet, la complexité de calcul pour XCiT augmente de manière linéaire avec le nombre de patches, tandis que pour les ViTs celle-ci augmente de manière quadratique.

Les XCiT conservent des connexions résiduelles et une normalisation de couche comme les ViT. Une autre différence importante réside dans l'inclusion d'opérations convolutives pour renforcer la capture des relations locales, ce qui manque aux ViT traditionnels. Ces convolutions sont intégrées après chaque couche d'attention pour mélanger les informations entre patches voisins.

Les couches feedforward suivent une structure similaire à celles des ViT, avec deux transformations linéaires et une activation non linéaire :

$$\text{FFN}(z) = \sigma(zW_1 + b_1)W_2 + b_2,$$

où $W_1 \in \mathbb{R}^{d \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d}$, et σ est souvent GELU.

En termes de performances, XCiT est plus efficace que ViT pour plusieurs tâches de vision, en particulier sur des jeux de données de taille modeste, où l'ajout d'opérations locales (via les convolutions) améliore la généralisation. De plus, la réduction de la complexité du mécanisme d'attention rend XCiT plus adapté pour des applications nécessitant une réponse rapide du modèle. En effet, on pourrait croire qu'en réduisant l'espace "clés - requêtes" en une matrice $d \times d$ plutôt qu'une matrice $N \times N$ on obtiendrait des performances bien moins correctes, néanmoins, dans le papier associé à XCiT (XCiT: Cross-Covariance Image Transformers Touvron et al. 2021) [9], il est montré que XCiT performe très bien sur certains datasets comme ImageNet-1k.

En conclusion, XCiT combine les forces des Transformers globaux comme ViT avec les avantages d'un mécanisme d'attention permettant de réduire la complexité des calculs, ce qui, permet nottament un entraînement plus rapide.

5.2.2 Architecture

On montre ici l'architecture d'un XCiT de manière plus imagée 11:

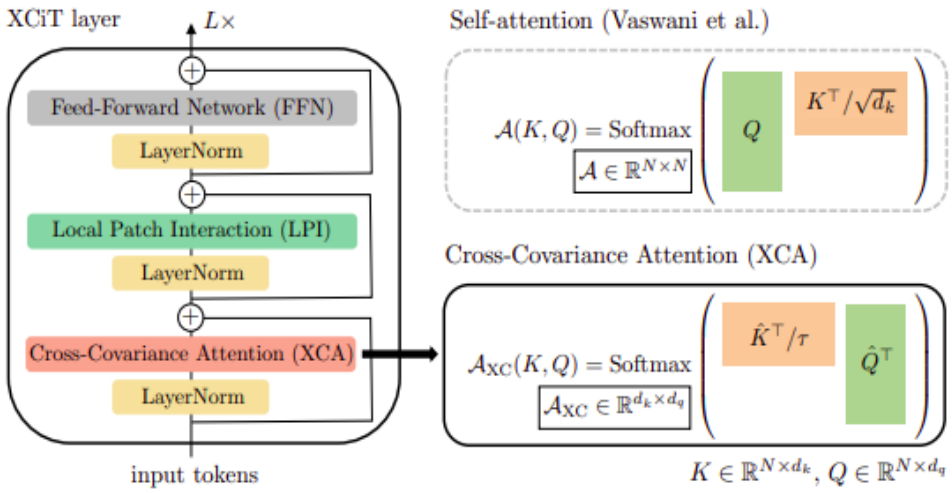

Figure 11: Architecture d'un XCiT

5.2.3 Visualisation

L'explicabilité du fonctionnement du modèle XCiT peut être visualisée de la même manière intuitive que les ViTs. En effet, en visualisant les résultats des différentes têtes d'attention, cette fois modifiées. Grâce à ces têtes d'attention on peut déterminer quelle partie du modèle (XCiT) est associée à quelle instance de l'image. Mais cette fois, on va simplement calculer la norme des vecteurs de la Cross-Attention sur l'autre dimension de celle-ci de sorte à avoir une carte de chaleur comme précédemment avec les ViTs:

$$Visualition(x,y) = ||crossw_{(x,y)}||$$

avec $x \in N_x$, $y \in N_y$ et $N_x \times N_y = N$ et $crossw_{(x,y)}$ le vecteur dans $XCA(Q,K,V)$ associé au token (x,y).

Le choix du nombre de têtes d'attention est donc important, néanmoins il est difficile d'estimer le nombre de têtes optimales.

On montre ici quelques exemples de visualisation de têtes d'attention XCiT pour le dataset ImageNet 12:

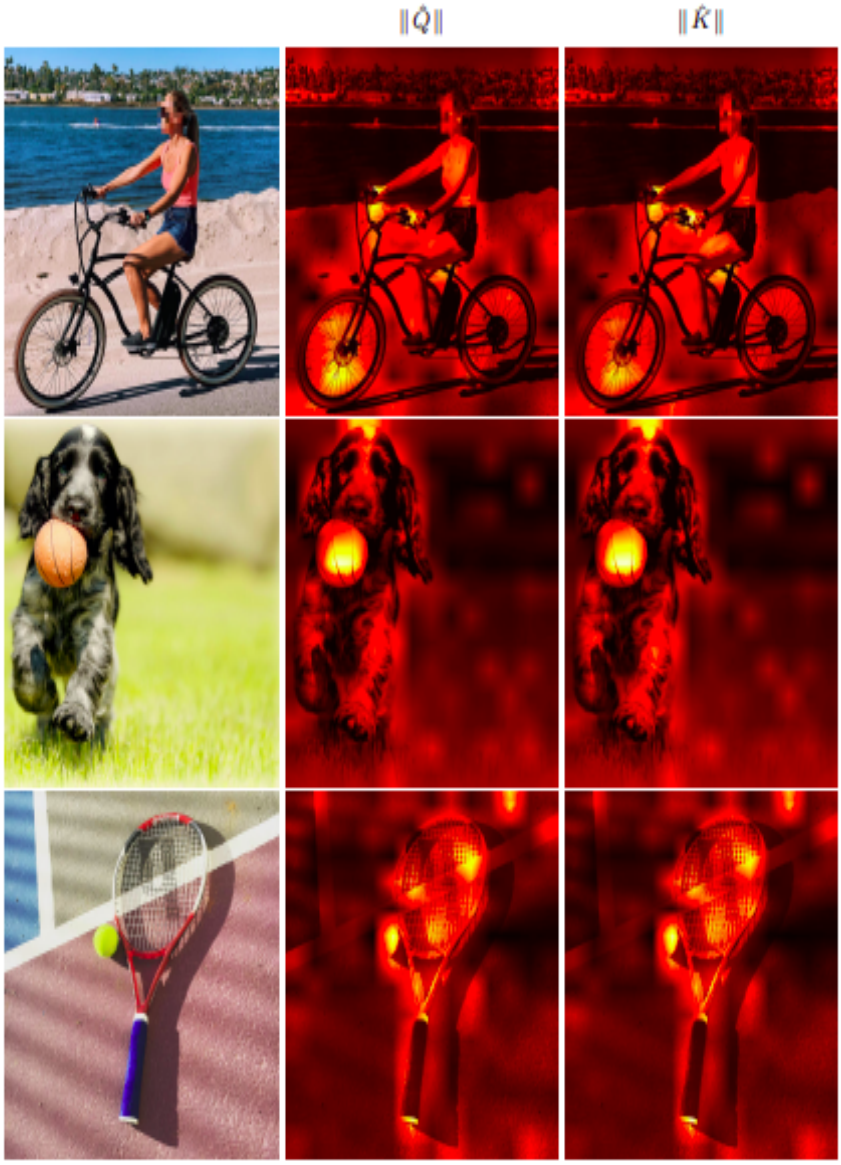

Figure 12: Visualisation des clés et requêtes pour XCiT

6 Entraînement d'un modèle XCiT avec DINO

Dans cette partie nous allons parler de la manière dont nous avons entraîné notre modèle XCiT avec le framework DINO. En effet, disposant d'un très grand nombre de données (10000 images Tiff dans le dataset PANDA), et le modèle XCiT avec DINO requiérant une grande mémoire et une grande capacité computationnelle, nous avons décidé d'utiliser une des ressources du BSC (Barcelona Supercomputing Center), à savoir MareNostrum 5 son supercalculateur, nous allons donc ainsi expliquer comment nous avons préparé les données et toutes les étapes nécessaires au déploiement d'un entraînement sur ce supercalculateur.

6.1 Préparation des données

Comme dit précédemment, nous disposons de 10000 images Tiff de grande dimension. Néanmoins, comme expliqué auparavant, le partenaire du projet a spécifié qu'il disposait d'images en 224×224. Ainsi, il a fallu "cropper" toutes les images Tiff du dataset. Néanmoins, il a également fallu mettre en place une politique de conservation des images, en effet, sur ces images Tiff, la majorité des images croppées sont des images de background (que du blanc) du fait que les images Tiff capturent un tissu d'une forme non rectangulaire. Nous avons donc mis en place une politique de conservation des images selon l'emptiness, les edges et l'homogénéité en fixant des valeurs de threshold.

Une fois ces données extraites, nous les avons stockées sur le supercalculateur en attendant de lancer l'entraînement.

Au final, on obtient un nombre de tiles (images croppées en 224×224) de 2852703.

6.2 Hyperparamètres pour DINO ?

Une fois les images obtenues, avant de lancer l'entraînement, il nous faut choisir les hyperparamètres du framework DINO. Pour cela, nous nous sommes principalement basés sur les valeurs par défaut du repository Github associé à DINO, néanmoins, il a fallu adapter certains hyperparamètres.

6.2.1 Dimension de sortie

La dimension de sortie de la tête finale de notre modèle est très importante puisqu'elle doit être assez grande pour représenter correctement la distribution de nos données sans être trop grande ce qui pourrait biaiser nos résultats en raison de redondance ou de difficultés à s'entraîner. Dans notre cas, la valeur conseillée par les auteurs du papier DINO est de 65536, néanmoins, ils opèrent sur des images bien plus grandes que les nôtres (ImageNet). Il n'est donc pas vraiment pertinent de définir comme sortie du modèle une dimension plus grande que le nombre de pixels dans nos images d'entrée (224×224).

C'est en effet une erreur que j'ai commise au début, les résultats n'étant pas très bons il a fallu inspecter les raisons de cet échec. En tatônnant et réfléchissant à la manière dont fonctionnait DINO il m'est ensuite apparu évident que laisser cette valeur par défaut était une erreur. Il a donc fallu déterminer quelle dimension de sortie choisir.

Nous avons donc testé plusieurs valeurs de dimension de sortie (en respectant tout de même la réduction de dimension par rapport à l'image d'entrée). Nous avons testé quatre valeurs: 1024, 2048, 4096 et 8192. Nous n'avons pas testé pour plus de valeurs de part le coût en temps et en calcul de l'entraînement pour chacune des valeurs. Après des tests sur des benchmarks (dont nous parlerons plus tard), il est apparu que la valeur optimale était 4096 (ce choix est empirique).

6.2.2 Batch size

Lors de l'entraînement une difficulté liée au batchsize est apparue. En effet, la fonction de coût de DINO est une somme de fonction de coût entre chaque vue augmentée des images, ainsi, le coût mémoire pour le stockage de celle-ci ainsi que le coût mémoire pour stocker les gradients est important, il a donc fallu adapter la taille du batch pour ne pas surcharger les différents GPUs

séléctionnés pour l'entraînement. Nous avons donc choisi un batchsize de 100, qui permettait un bon ratio rapidité d'entraînement et coût en mémoire.

6.2.3 Nombre de crops globales et locales

Il a également fallu choisir le nombre de vues augmentées pour la fonction de coût de DINO. En effet, les valeurs par défaut pour celles-ci sont de 2 vues globales en 224×224 pour le teacher, et 8 vues locales en 96×96 pour le student (le student prend également les vues globales dans son forward). Néanmoins, dans le papier de DINO (auquel sont associées les valeurs par défaut), les images sont bien plus grandes, il était donc inutile dans notre cas de garder autant de vues. Nous avons donc choisi de sélectionner 1 vue globale pour le teacher et 5 vues locales.

6.3 Hyperparamètres du modèle XCiT

Pour les hyperparamètres associés à XCiT il nous fallait en réalité chosir quel type de modèle XCiT choisir parmi les différents disponibles (inutile de créer une structure XCiT par nous mêmes alors que les infrastructures disponibles sur Github ont fait leur preuve). Il en existe plusieurs mais on peut les résumer en trois catégories: XCiT "small", "medium" ou "large". Notre partenaire nous a indiqué vouloir un XCiT "medium", nous avons donc choisi celui-ci dont voici la structure ci-après 13:

```
@register_model
def xcit_medium_24_p16(pretrained=False, **kwargs):
    model = XCiT(
        patch_size=16, embed_dim=512, depth=24, num_heads=8, mlp_ratio=4, qkv_bias=True,
        norm_layer=partial(nn.LayerNorm, eps=1e-6), eta=1e-5, tokens_norm=True, **kwargs)
    model.default_cfg = _cfg()
    return model
```

Figure 13: Hyperparamètres modèle "xcit medium"

6.4 Préparer l'Entraînement

Pour lancer l'entraînement, il a fallu se familiariser avec plusieurs ressources, les voici ci-dessous ainsi que le pipeline pour lancer l'entraînement.

6.4.1 BSC Supercomputer

MareNostrum 5, le supercalculateur du Barcelona Supercomputing Center (BSC), est l'une des infrastructures de calcul les plus avancées en Europe. Conçu pour atteindre une puissance de calcul exaflopique, il intègre des technologies de pointe combinant processeurs haute performance et accélérateurs GPU pour des applications intensives en calcul et en données.

Destiné à soutenir des recherches dans des domaines variés comme la médecine, le climat, l'énergie ou l'intelligence artificielle (dans mon cas l'IA au service de la médecine), MareNostrum 5 est financé par l'Union Européenne et l'Espagne dans le cadre de l'initiative EuroHPC.

MareNostrum 5 offre un accès à la communauté scientifique mondiale, consolidant le rôle du BSC comme un leader international en calcul haute performance.

J'ai donc eu la chance, dans le cadre de mon stage, de pouvoir utiliser cette ressource européenne.

6.4.2 Singularity

Singularity est une plateforme de conteneurisation open source conçue spécifiquement pour les environnements de calcul haute performance (HPC). C'est dans ce contexte d'HPC que l'option Singularity a été préférée par les membres de mon équipe.

Les conteneurs Singularity encapsulent les dépendances logicielles et les environnements nécessaires, garantissant une reproductibilité des expériences et des analyses, même sur des systèmes hétérogènes,

de la même manière que Docker. Singularity est particulièrement adapté aux chercheurs et ingénieurs en sciences computationnelles, car il est compatible avec les gestionnaires de tâches des clusters HPC.

Avec sa simplicité d'utilisation, Singularity est devenu un outil essentiel dans le domaine du calcul scientifique.

Il a donc fallu créer un fichier .def (à l'image d'un Dockerfile) pour créer une image Singularity permettant de lancer mon script Python sur le Supercalculateur. Voici le fichier en question avec toutes les bibliothèques Python nécessaires 14:

```
BootStrap: docker
From: nvidia/cuda:12.3.2-cudnn9-runtime-ubuntu22.04

%post
    # Install Python 3.10
    apt update -y
    apt install -y software-properties-common
    add-apt-repository ppa:deadsnakes/ppa -y
    apt update -y
    apt-get install -y git
    apt install -y curl
    apt install -y python3.10
    update-alternatives --install /usr/bin/python3 python3 /usr/bin/python3.10 1
    update-alternatives --install /usr/bin/python python /usr/bin/python3.10 1
    apt install -y python3.10-venv python3.10-dev
    curl -Ss https://bootstrap.pypa.io/get-pip.py | python3.10
    apt-get clean
    rm -rf /var/lib/apt/lists/

    # Install
    # torch        2.4.1+cu121
    # torchvision  0.19.1+cu121
    # numpy        2.0.2
    # pillow       10.4.0
    # datasets     3.0.0
    # timm         1.0.9
    # tifffile     2024.8.24
    # imagecodecs  2024.6.1
    # submitit     1.5.2
    # hydra-core   1.3.2
    # omegaconf    2.3.0
    # pywt         1.8.0
    pip install --upgrade pip
    pip3 install torch==2.4.1+cu121 torchvision==0.19.1+cu121 --index-url https://download.pytorch.org/whl/cu121
    pip3 install numpy==2.0.2 pillow==10.4.0 datasets==3.0.0 timm==1.0.9 tifffile==2024.8.24 imagecodecs==2024.6.1

%environment
# Set any environment variables if required

%runscript
# Add a runscript if needed

%labels
   Author Emile Vaysse
   Institution Barcelona Supercomputing Center


```

Figure 14: Fichier .def (une adaptation du DockerFile)

6.4.3 Lancement de l'entraînement

Pour lancer l'entraînement il a fallu définir un script .sh pour pouvoir lancer de manière distribuée le script Python associé à l'entraînement de notre modèle XCiT. Pour une rapidité d'entraînement nous avons choisi d'utiliser 6 nodes ainsi que 4 GPUs par node. Voici le script .sh ainsi que les commandes SBATCH (variables pour le cluster) associées 15:

6.5 Suivi de l'entraînement

Voici ci-après l'évolution de la fonction de coût associée à DINO pour notre modèle XCiT 16:

```
#SBATCH -J test-mn5      # Job name
#SBATCH -o slurm_output/out.txt       # Name of stdout output file(%j expands to jobId)
#SBATCH -e slurm_output/err.txt       # Name of stderr output file(%j expands to jobId)
#SBATCH --account bsc70              # Account name
#SBATCH -t 22:00:00         # Run time (hh:mm:ss)
#SBATCH --qos=acc_bsccs
#SBATCH --gres=gpu:4  # Request 4 GPUs per node
#SBATCH --cpus-per-task=80
#SBATCH --ntasks-per-node=1
#SBATCH --nodes=6  # Request 4 nodes

echo COMMIT_TAG=$COMMIT_TAG

export SLURM_CPU_BIND=none
export SRUN_CPUS_PER_TASK=${SLURM_CPUS_PER_TASK}
export USE_SYSTEM_NCCL=1

export PATH=/apps/ACC/CUDNN/9.1.0/cuda12/lib:$PATH

# Setting important distributed training variables
export GPUS_PER_NODE=4
export NNODES=$SLURM_NNODES
export NUM_PROCESSES=$(expr $NNODES \* $GPUS_PER_NODE)
export MASTER_ADDR=$(scontrol show hostnames $SLURM_JOB_NODELIST | head -n 1)
export MASTER_PORT=6000  # Change this port if necessary

echo "START TIME: $(date)"

# Load necessary modules
module purge
module load cuda/12.1 mkl/2024.0 nvidia-hpc-sdk/24.3 openblas/0.3.27-gcc intel/2024.0 cudnn/9.1.0-cuda12
module load singularity

# Set paths for CUDA, CUDNN, and other libraries
export CUDA_NVCC_EXECUTABLE="/apps/ACC/NVIDIA-HPC-SDK/24.3/Linux_x86_64/24.3/cuda/12.3/bin/nvcc"
export CUDA_HOME="/apps/ACC/NVIDIA-HPC-SDK/24.3/Linux_x86_64/24.3/cuda/12.3"
export CUDNN_INCLUDE_PATH="/apps/ACC/CUDNN/9.1.0/cuda12/include"
export CUDNN_LIBRARY_PATH="/apps/ACC/CUDNN/9.1.0/cuda12/lib"

# Configure compilation
export USE_CUDA=1 USE_CUDNN=1 USE_MKLDNN=0 USE_MKL=1 USE_TENSORRT=1 USE_XPU=0 MKL_THREADING=OMP

# Use srun for distributed execution
srun singularity exec --nv /home/bsc/bsc910833/dino_xcit/container_histology_dino_xcit3.sif bash -c "
    python -m torch.distributed.run \
        --nproc_per_node=$GPUS_PER_NODE \
        --nnodes=$SLURM_NNODES \
        --rdzv_endpoint=$MASTER_ADDR:$MASTER_PORT \
        --rdzv_backend=c10d \
        dino_train_4096_2.py
"

echo "END TIME: $(date)"
```

Figure 15: Script sh pour lancer l'entraînement sur le supercalculateur

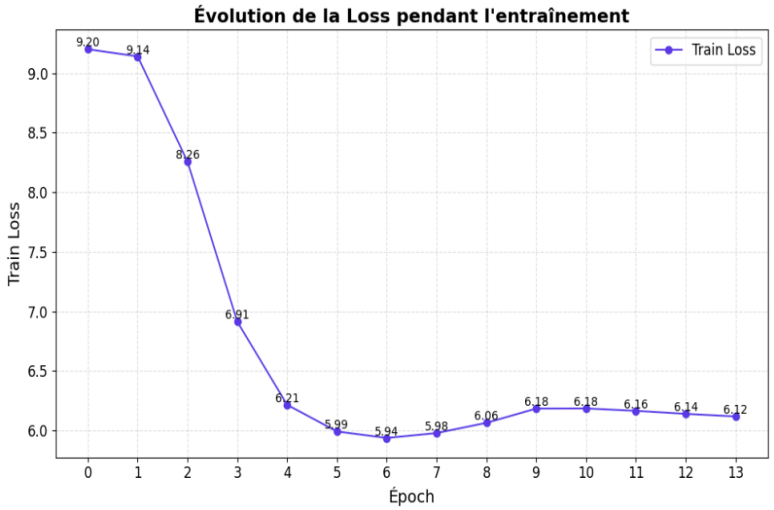

Figure 16: Fonction de coût de DINO lors de l'entraînement

On voit ainsi une diminution globale de la fonction de coût, néanmoins, on remarque parfois une augmentation de celle-ci notamment au bout d'une dizaine d'epochs. Mais selon les auteurs du papier DINO (via l'onglet "issues" du repository Github de DINO), l'augmentation de la fonction de coût n'est pas forcément un mauvais signe, en effet, l'interprétation d'une telle courbe est diffcile en raison de la nature même de cette fonction de coût. Une augmentation de celle-ci pourrait être bénéfique et conduire, à terme, à un meilleur comportement du modèle final.

Il reste désormais à tester si notre modèle entraîné de manière auto-supervisé a réussi à capturer les bonnes informations dans nos images.

7 Evaluation sur des downstreams tasks

Une fois l'entraînement de notre modèle XCiT effectué sur le super calculateur, que nous appelons Bony, il nous faut tester ce modèle. Pour cela, comme décrit précédemment dans ce rapport, on va tester ce modèle sur plusieurs benchmarks. A savoir un subset du dataset PANDA sur lequel on a effectué l'entraînement non-supervisé (ici on pourrait questionner la pertinence de tester notre modèle sur le même dataset que l'entraînement mais comme l'entraînement n'était pas supervisé cela reste pertinent), le dataset DeepGleason et enfin le dataset SICAPv2.

Néanmoins, comment tester notre modèle sur ces benchmarks ? Pour cela, il faut considérer notre modèle XCiT entraîné sur PANDA de manière auto-supervisé comme un encodeur. Un encodeur sur lequel on va venir greffer un décodeur qui lui effectuera une tâche de classification ou de segmentation. Pour la classification, les labels étant disponibles, un label sera affecté à chaque image 224x224. Comme expliqué précédemment, pour PANDA, on affectera le label le plus élevé en terme d'échelle Gleason disponible pour cette image. Pour la segmentation on tentera de retrouver une représentation plus simple de l'image mise en entrée du modèle XCiT en fonction de ce qui est disponible.

7.1 Décodeur pour le benchmark subset de PANDA

Comme expliqué précdemment, pour le benchmark associé à PANDA nous avons accès à des labels qui sont contenus dans des images de segmentation (masks) associées à chaque image du dataset. On a donc un label pour chaque image du benchmark. On a également un embedding de dimension 4096 en sortie de notre encodeur XCiT. La stratégie que nous avons choisi d'appliquer est de greffer un décodeur en sortie du modèle XCiT.

Ce décodeur est un MLP (Multi Layer Perceptron), dont la structure est 17:

```python
class XCiTWithMLPClassifier_4096(nn.Module):
    def __init__(self, checkpoint_path, num_classes=5, hidden_dims=[7192, 2048, 512, 128]):
        super(XCiTWithMLPClassifier_4096, self).__init__()

        # Charger le modèle pré-entraîné depuis le checkpoint
        # self.model = torch.hub.load('facebookresearch/dino:main', 'dino_xcit_medium_24_p8')
        model = xcit_medium_24_p16(pretrained=False)

        d = torch.load('./checkpoint_4096.pth')

        model = MultiCropWrapper(
        model,
        DINOHead(512, 4096, False),
    )
        model.load_state_dict(d['teacher'])
        self.model = model

        for param in self.model.parameters():
            param.requires_grad = False

        num_features = 4096

        layers = []
        input_dim = num_features

        layers.append(nn.Linear(input_dim, hidden_dims[0]))
        layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_dims[0], hidden_dims[1]))
        layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_dims[1], hidden_dims[2]))
        layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_dims[2], hidden_dims[3]))
        layers.append(nn.ReLU())

        # Ajouter la dernière couche pour les classes de sortie
        layers.append(nn.Linear(hidden_dims[3], num_classes))
        layers.append(nn.Softmax(dim=1))

        self.classifier = nn.Sequential(*layers)

    def forward(self, x):
        x = self.model(x)  # Passage dans le modèle pré-entraîné
        return self.classifier(x)  # Passage dans le MLP
```

Figure 17: Code du classifieur greffé à l'encodeur

Il est à noter que plusieurs architectures de décodeur ont été testées pour tenter de se rapprocher du décodeur optimale. Pour cela nous avons principalement testé plusieurs hyperparamètres tout en observant les phénomènes d'overfitting ou d'underfitting de sorte à optimiser au mieux l'entraînement de ce décodeur.

Le but de ce décodeur était de pouvoir tenter de savoir si le modèle entraîné sur le super-calculateur était capable de généraliser l'information. Ainsi, il nous était possible de déterminer si le modèle XCiT pouvait être envoyé au partenaire, qui lui, fait la même chose mais avec un dataset privé qui leur appartient.

Voici les courbes d'entraînement pour le décodeur décrit plus haut, il est à noter que le train-val-test split a été fait en (0.8 0.1 0.1) et que les paramètres de l'encodeur (XCiT) ont été "freezés" (ces paramètres ne sont pas actualisés pendant l'entraînement):

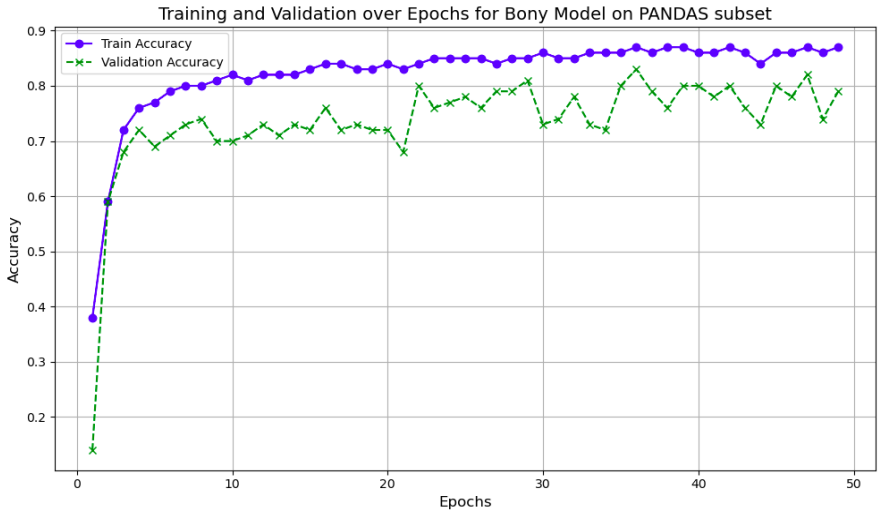

Figure 18: Train accuracy for Bony model on PANDAS subset

Avec 18 on observe une évolution jointe des deux courbes, on observe pas vraiment d'overfitting, il aurait peut être était une bonne idée d'essayer de comlexifier le décodeur. En outre, il aurait pu être sage d'utiliser du early-stopping sur notre entraînement mais il a été choisi de définir un nombre d'epochs fixe de sorte à observer l'entraînement sur la durée. Ici il est clair que l'entraînement était dans une phase relativement constante sans réelle amélioration. Si cela n'avait pas été le cas, nous aurions continué l'entraînement.

Sur 18 on observe un entraînement convenable et une performance sur le test de 0.81 ce qui est une bonne performance sur ce benchmark comparé à notre modèle Baseline (en effet nous avons utilisé un modèle ViT appelé "Hibou" disponible sur Huggingface entraîné sur un dataset privé de 1.2 milliards de "tiles", qui sont des crops d'images Tiff de dimension 224x224 (comme les nôtres)). Notre modèle est donc entraîné avec moins d'images (2.5 millions d'images, soit un ratio de $\frac{2.5}{1200}$. Mais en comparaison sa performance est compétitive, voici ci-après les courbes d'apprentissage du décodeur avec un encodeur Hibou 19:

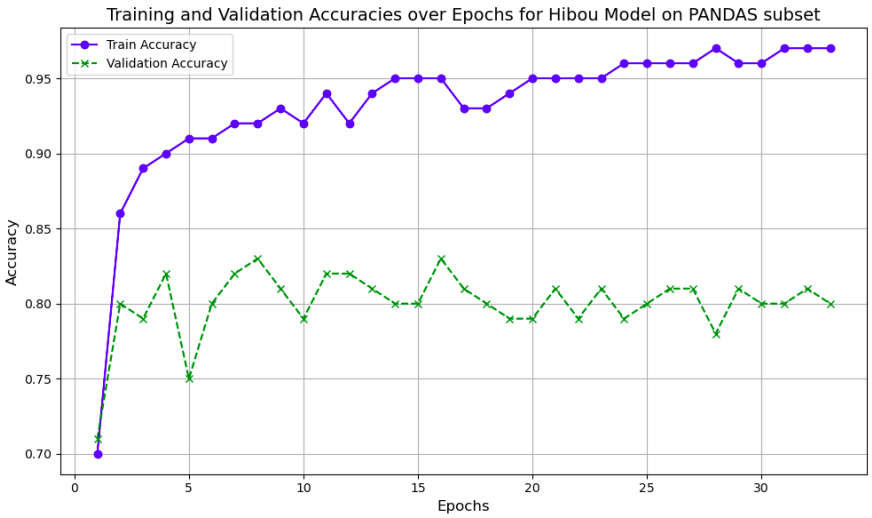

Figure 19: Train accuracy for Hibou model on PANDAS subset

Avec 19 on observe une courbe de validation qui n'augmente pas vraiment conjointement avec le Train. En effet, l'apprentissage semble être stoppé, un écart se creuse entre le Train et la

Validation. Néanmoins les performances restent convenables avec une accuracy autour de 80%.

On pourrait se demander pourquoi Hibou n'écrase pas notre modèle du fait de son entraînement massif comparé au nôtre. En réalité, Hibou est entraîné sur un grand nombre d'organes, et pas que de la Prostate. Ceci donne un avantage à notre modèle.

7.2 Décodeur pour le benchmark DeepGleason

Dans le cas de DeepGleason, nous avons accès aux segmentations (masks). Nous avons donc choisi une évaluation différente que pour PANDA en considérant comme label le mask associé au crop de chaque image du dataset DeepGleason. Ainsi, l'encodeur XCiT générera un embedding, ensuite, il passera dans le décodeur qui est une Convolution 2D Transposée, de sorte à avoir un décodeur qui augmente la dimension de l'embedding de base jusqu'à avoir une image en 224x224. Elle sera ensuite comparée au mask obtenu précédemment par une MSE (Mean Squared Error), comme résumé dans le diagramme 20.

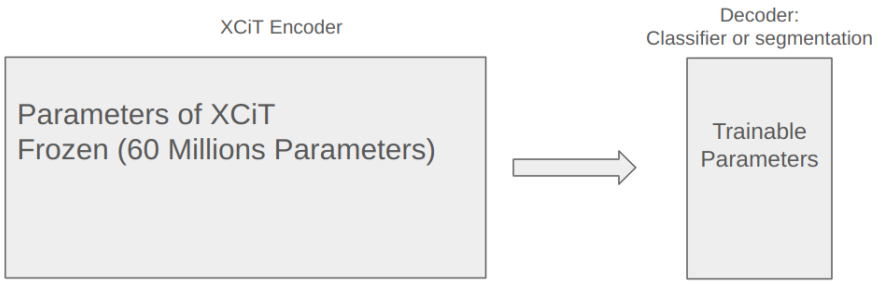

Figure 20: Diagramme encodeur - décodeur

Voici la structure du décodeur en question 21:

Voici les courbes d'entraînement 22 pour le décodeur décrit plus haut, il est à noter que le train-val-test split a été fait en (0.8 0.1 0.1) et que les paramètres de l'encodeur (XCiT) ont été "freezés" (ces paramètres ne sont pas actualisés pendant l'entraînement):

```python
class XCiTWithDecoder_4096(nn.Module):
    def __init__(self, checkpoint_path, num_classes=5, output_channels=1):
        super(XCiTWithDecoder_4096, self).__init__()
        model = xcit_medium_24_p16(pretrained=False)

        d = torch.load('./checkpoint_4096.pth')

        model = MultiCropWrapper(
        model,
        DINOHead(512, 4096, False),
    )
        model.load_state_dict(d['teacher'])

        self.model = model

        for param in self.model.parameters():
            param.requires_grad = False

        num_features = 4096

        self.fc = nn.Linear(num_features, 256 * 7 * 7)

        self.upsample = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.ConvTranspose2d(16, output_channels, kernel_size=3, stride=2, padding=1, output_padding=1),

            nn.Sigmoid()  # Optional: normalize to [0, 1] for image data
        )

    def forward(self, x):
        x = self.model(x)  # Passage dans le modèle pré-entraîné
        x = self.fc(x)
        x = x.view(x.size(0), 256, 7, 7)
        return self.upsample(x)  # Passage dans le MLP
```

Figure 21: Code du décodeur (sgmentation) pour DeepGleason greffé à l'encodeur

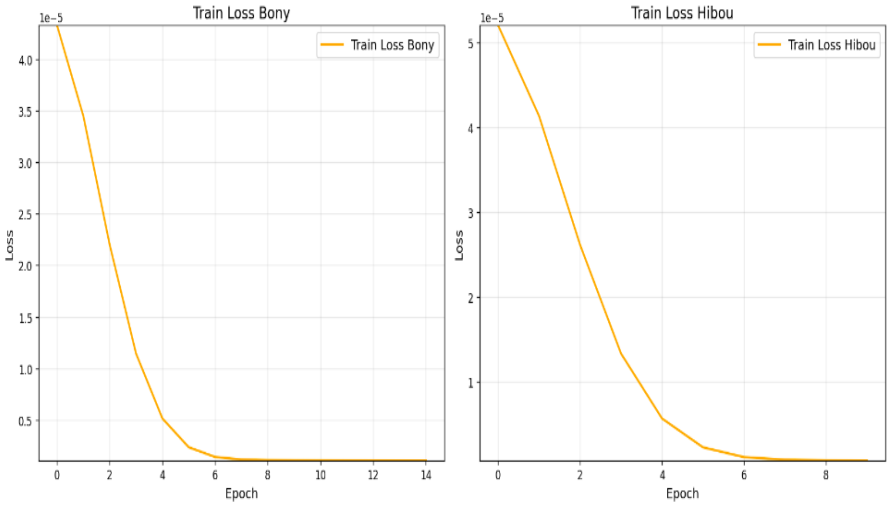

Figure 22: Fonction de coût d'entraînement (segmentation) pour Bony et Hibou pour DeepGleason

Sur la figure 22 on observe un entraînement qui semble stable et consistent étant donné que la Loss diminue bien tant pour Bony que pour Hibou. Notre modèle Baseline Hibou a de meilleurs résultats d'entraînement que Bony (Entraînement Bony à la fin: 1.07e-6 et Hibou: 7.31e-7) comme on peut le voir sur la figure 23:

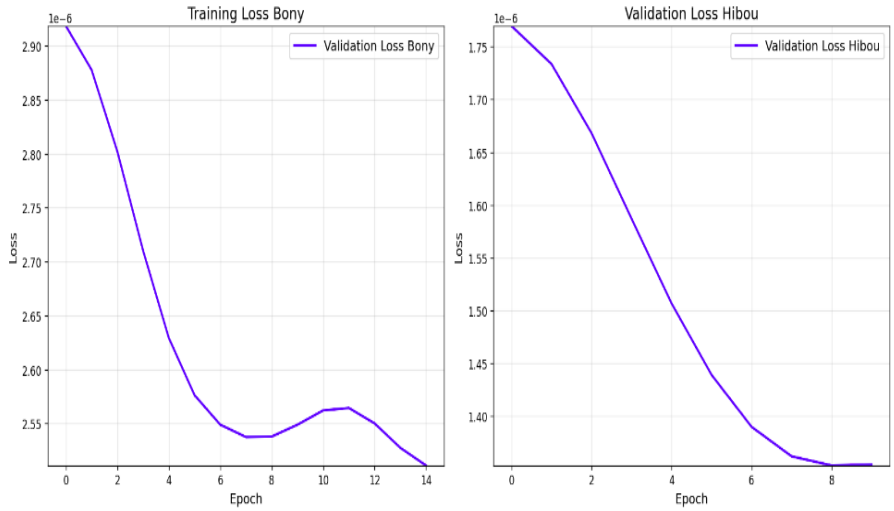

Figure 23: Fonction de coût de validation (segmentation) pour Bony et Hibou pour DeepGleason

La performance sur la validation atteint 10% de la valeur moyenne des pixels ce qui est une très bonne performance sur ce benchmark pour Bony. Notre modèle est donc entraîné avec moins d'images (avec un ration de $\frac{2.5}{1200}$) mais la performance de Hibou (1.4×10^{-6}) n'est pas bien meilleure que celle de notre modèle: 2.9×10^{-6} quand on sait que les images sont normalisées entre 0 et 1.

7.3 Décodeur pour le benchmark SICAPv2

Dans le cas de SICAPv2, nous n'avons pas accès aux segmentations (masks). Nous avons donc choisi une évaluation différente que pour les autres datasets.

En effet, nous considérons comme label, l'image d'entrée, mais "blurrée" de manière Gaussienne. On peut ainsi observer si le modèle a bien généralisé l'information contenue dans l'image.

Néanmoins, on "blurre" l'image puisque le décodeur en question est un décodeur Convolutif et que les images d'Histopathologie sont des images très peu continues et difficile à reconstruire de bout en bout.

Il nous faut donc "blurrer" l'image pour tout de même observer si l'encodeur a bien généralisé l'entraînement.

Ainsi, l'encodeur XCiT générera un embedding, ensuite, il passera dans le décodeur qui est une Convolution 2D Transposée.

Nous avons ensuite un décodeur qui augmente la dimension de l'embedding de base jusqu'à avoir une image en 224x224 qui sera ensuite comparée à la même image d'entrée mais "blurrée" avec une MSE (Mean Squared Error).

Voici la structure du décodeur en question 24:

```python
class XCiTWithDecoder_4096(nn.Module):
    def __init__(self, checkpoint_path, num_classes=5, output_channels=3):
        super(XCiTWithDecoder_4096, self).__init__()

        model = xcit_medium_24_p16(pretrained=False)

        d = torch.load('../checkpoint_4096.pth')

        model = MultiCropWrapper(
        model,
        DINOHead(512, 4096, False),
    )
        model.load_state_dict(d['teacher'])
        self.model = model

        for param in self.model.parameters():
            param.requires_grad = False

        num_features = 4096

        self.fc = nn.Linear(num_features, 256 * 7 * 7)
        self.upsample = nn.Sequential(
            nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1, output_padding=1),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.ConvTranspose2d(16, output_channels, kernel_size=3, stride=2, padding=1, output_padding=1),

            nn.Sigmoid()  # Optional: normalize to [0, 1] for image data
        )

    def forward(self, x):
        x = self.model(x)  # Passage dans le modèle pré-entraîné
        x = self.fc(x)
        x = x.view(x.size(0), 256, 7, 7)
        return self.upsample(x)  # Passage dans le MLP
```

Figure 24: Code du décodeur (segmentation) pour SicapV2 greffé à l'encodeur

Voici les courbes d'entraînement pour le décodeur décrit plus haut 25, il est à noter que le train-val-test split a été fait en (0.8 0.1 0.1) et que les paramètres de l'encodeur (XCiT) ont été *freezés* (ces paramètres ne sont pas actualisés pendant l'entraînement):

Sur la figure 25 on observe un entraînement qui semble bon étant donné que la Loss diminue bien tant pour Bony que pour Hibou avec une performance autour de 0.00082 de la valeur moyenne des pixels pour Bony ce qui est une très bonne performance sur ce benchmark. Notre modèle Baseline Hibou lui obtient de meilleurs résultats pour l'entraînement. On résume les performances sur la validation ci-après. 26:

Sur ce benchmark Hibou est meilleur que notre modèle XCiT pendant l'entraînement mais pas lors de la validation. Ceci montre peut être une meilleure capacité de Bony à généraliser l'information contenue dans chaque image, ce qui est une très bonne performance pour Bony étant donné que celui-ci est entraîné sur moins d'images que Hibou. Néanmoins, il faut rappeler que Hibou a une dimension de sortie de seulement 768 alors que Bony a une dimension de sortie de 4096, ce qui peut expliquer en partie ces résultats.

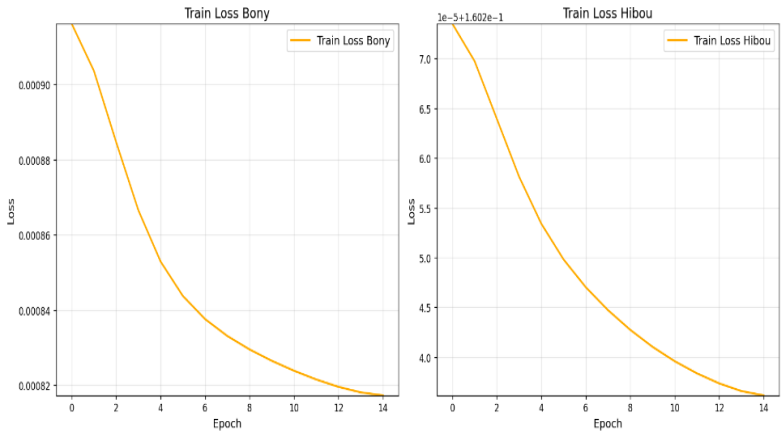

Figure 25: Fonctions de coût d'entraînement (segmentation) pour Bony et Hibou pour SICAPv2

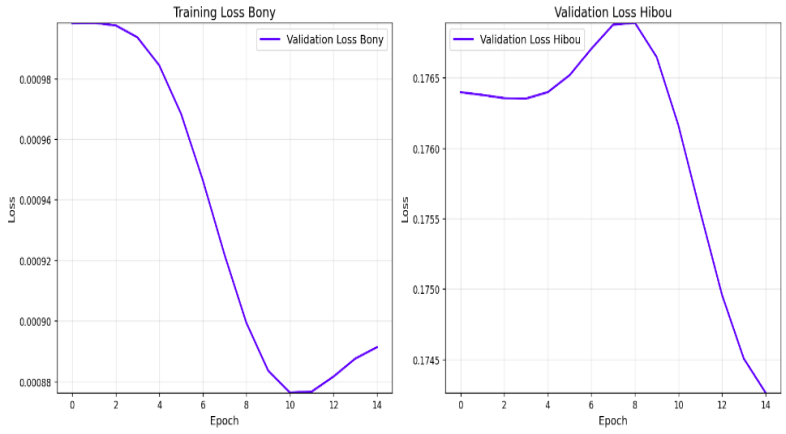

Figure 26: Fonctions de coût de validation (segmentation) pour Bony et Hibou pour SICAPv2

7.4 Conclusion

Ces résultats semblent donner Hibou comme un meilleur modèle que notre XCiT sauf sur un dataset, les raisons de cet écart de performance entre Hibou et XCiT sur ce benchmark (alors qu'ils sont "comparables" sur les autres benchmarks) ne sont pas connues exactement mais notre hypothèse est que, étant donné que Hibou n'est pas entraîné principalement sur des images d'Histopathologie de Prostate et que sa dimension de sortie soit de 768, il est possible qu'il ait eu du mal à segmenter ces images de Prostate du benchmark SICAPv2.

8 Récapitulatif des résultats

On récapitule les différents résultats de nos meilleurs modèles pour chaque benchmark:

Model	PANDA test subset (Accuracy) ↑	DeepGleason (MSE) ↓	SICAPv2 (MSE) ↓
Bony	81.2%	$2.934e-06$	**0.0008**
Hibou	**83.1**%	$1.455e-06$	0.10
Histoencoder	81.6%	**1.003e − 06**	–

"Histoencoder" est l'un des modèles baseline que nous avons utilisé (comme dit précédemment) et qui suit une méthode similaire à la nôtre. Il est issu du papier (Pohjonen et al. 2024) [10].

Sur le benchmark PANDA test subset, qui mesure la précision, le modèle Hibou atteint la meilleure performance avec un score de 83.1%, surpassant Bony (81.2%) et Histoencoder (81.6%). Ce résultat met en évidence la capacité de Hibou à généraliser efficacement sur ce dataset. Le modèle Histoencoder, bien qu'il dépasse Bony, reste légèrement en retrait, ce qui suggère qu'il est robuste mais pas optimisé spécifiquement pour cette tâche, ce qui peut être étonnant étant donné que Histoencoder est lui entraîné sur des images de Prostate. Néanmoins, cette observation est à prendre avec des pincettes, ce test ne reste qu'un test assez limité en terme de volume.

Pour le benchmark DeepGleason, où une faible MSE est recherchée, Histoencoder se démarque en obtenant la meilleure performance avec une MSE de 1.003e-06. Hibou suit avec une MSE de 1.455e-06, tandis que Bony montre des résultats nettement inférieurs avec une MSE de 2.934e-06. Cela indique que Histoencoder est particulièrement adapté à la tâche de segmentation, grâce à sa capacité à extraire des représentations précises. En revanche, les performances inférieures de Bony reflètent des lacunes dans sa capacité à capturer les subtilités des données du benchmark DeepGleason. Néanmoins, Bony a des résultats qui restent compétitifs.

En ce qui concerne SICAPv2, Bony obtient les meilleurs résultats avec une MSE de 0.0008, tandis que Hibou affiche une MSE bien plus élevée de 0.10. Le modèle Histoencoder n'a pas été testé sur ce benchmark. La supériorité de Bony sur SICAPv2 peut être attribuée à une architecture particulièrement bien adaptée aux caractéristiques spécifiques de ce dataset. À l'inverse, les résultats relativement faibles de Hibou sur ce benchmark montrent une certaine fragilité de ce modèle. Ceci est certainement dû au fait que Hibou a un embedding de sortie de seulement 768, rendant les tâches de segmentation plus complexes.

Globalement, chaque modèle présente des forces distinctes. Hibou se distingue par sa polyvalence, obtenant la meilleure précision sur PANDA et restant compétitif sur DeepGleason. Histoencoder, quant à lui, excelle dans les tâches de segmentation comme DeepGleason. Enfin, Bony montre une spécialisation notable sur SICAPv2, et ses performances globales restent compétitives.

Pour valider ces résultats, il faudrait conduire des tests plus complets, avec un plus grand nombre d'images notamment. Ainsi, ces résultats sont à prendre avec des pincettes.

9 Idées d'amélioration

Nous avons donc lors de ce rapport explorer une méthode d'apprentissage et de test dans un but précis. Dans cette partie, nous allons tenter d'exposer des idées d'amélioration de notre processus d'apprentissage d'un modèle XCiT.

9.1 DINOv2

Le framework DINOv2 (Jegou et al.) [8] représente une avancée significative dans le domaine de l'apprentissage auto-supervisé, améliorant les fondations posées par DINO. Tandis que DINO repose sur la distillation auto-supervisée pour apprendre des représentations visuelles robustes, DINOv2 affine cette méthode en introduisant une optimisation sophistiquée de la construction du jeu de données et de la fonction de perte.

9.1.1 Principes Fondamentaux de DINO

Comme introduit précédemment, DINO utilise une configuration enseignant-étudiant où l'étudiant apprend à reproduire les sorties de l'enseignant pour des vues augmentées d'une même image. La fonction de perte est celle introduite plus tôt dans ce rapport:

$$\mathcal{L} = \sum_i D_{\mathrm{KL}}(p_{\mathrm{teacher}}(x_i) \| p_{\mathrm{student}}(x_i)),$$

9.1.2 Avancées Introduites par DINOv2

DINOv2 conserve la distillation auto-supervisée mais apporte plusieurs améliorations:

- **Construction des Jeux de Données** : DINOv2 introduit un pipeline automatique pour filtrer les données non conformes. Contrairement à des approches antérieures qui s'appuyaient sur des métadonnées ou des encodeurs pré-entraînés, DINOv2 utilise des similarités visuelles directes, inspirées des pipelines de curation textuelle (Wenzek et al., 2020) [11]. Cette méthode garantit une meilleure qualité des données tout en restant auto-supervisée.

- **Optimisation de la Fonction de Perte** : DINOv2 introduit une perte discrimante, tout comme DINO. Mais DINOv2 introduit une autre fonction de coût, celle de iBot:

$$\mathcal{L}_{\mathrm{BOT}} = -\sum_i p_t^{(i)} \log p_s^{(i)},$$

  où :

  - $p_t^{(i)}$ représente la probabilité issue du modèle enseignant pour la classe i,
  - $p_s^{(i)}$ représente la probabilité issue du modèle étudiant pour la même classe i.
  - i représente l'indice des patchs en entrée qui sont masqués pour le student mais pas pour le teacher (voir (Zhou et al., 2022a) [12])

  L'utilisation de ces deux fonctions de coût permet de trouver des combinaisons plus riches grâce à iBOT et une bonne généralisation grâce à la fonction de coût de DINO déjà donnée précédemment.

- **Représentations Prêtes à l'Emploi** : Contrairement à DINO, dont les représentations nécessitent souvent un fine-tuning pour exceller sur des *downstream tasks*, DINOv2 produit des représentations directement utilisables sur des benchmarks comme ImageNet, avec des performances compétitives sans supervision supplémentaire.

9.1.3 Comparaison avec DINO et Perspectives

DINOv2 se distingue de DINO par sa capacité à tirer parti des données à grande échelle sans compromettre la qualité des représentations apprises. En effet, alors que DINO peut montrer des limites lorsqu'il est confronté à des données non équilibrées ou bruitées, DINOv2 surmonte ces défis grâce à son pipeline de curation de données robuste.

DINOv2 est donc une potentielle amélioration pour notre projet, de plus, dans notre contexte histopathologique, la curation des données d'entrées pourrait permettre d'entraîner sur plus d'images et une meilleure compréhension de notre modèle des images d'histopathologie. En effet, ces images présentent de grande similarité, néanmoins, le défi serait de trouver des datasets assez conséquents pour entraîner notre modèle DINOv2, ce qui n'était pas le cas dans notre contexte.

9.2 Décomposition en ondelettes

9.2.1 Motivations

Une autre idée d'amélioration possible m'est venue durant ce stage. Comme dit précédemment, les images d'Histopathologie sont des images très discontinues, bruitées et qui semblent se ressembler beaucoup. Ainsi, appliquer un filtre à ces images pourraient peut être permettre d'abstraire l'information des images, et ainsi, permettre un entraînement plus stable et peut être même plus efficace. C'est donc pour cela que l'idée de la décomposition en ondelettes en amont du forward dans notre modèle XCiT pourrait selon moi être une bonne idée.

9.2.2 Rappels sur la décomposition en ondelettes 3D

La décomposition en ondelettes 3D est une méthode adaptée pour analyser des données volumétriques, comme les images $224 \times 224 \times 3$, en extrayant des informations localisées à différentes échelles spatiales.

Les ondelettes sont des fonctions oscillantes localisées dans le temps et l'espace, utilisées pour décomposer un signal $f(x, y, z)$ en différentes échelles et orientations. La transformation en ondelettes 3D est définie par :

$$W_\psi f(j, \theta, x, y, z) = f * \psi_{j,\theta}(x, y, z),$$

où $\psi_{j,\theta}$ est une ondelette 3D avec :

- j : une échelle définissant la résolution spatiale,
- θ : une orientation spatiale spécifique,
- $*$: l'opérateur de convolution 3D.

Les ondelettes 3D courantes incluent les ondelettes de Morlet et de Haar, adaptées pour capturer les variations directionnelles.

9.2.3 2. Scattering 3D : Extension Invariante

Le scattering 3D est une méthode associée à la décomposition en ondelettes qui produit des représentations invariantes aux transformations (translation, rotation, etc.), ainsi, une image d'Histopathologie sera invariante dans le domaine de ses coefficients d'ondelettes ce qui permet une meilleure généralisation.

Étape 1 : Décomposition en Ondelettes On applique une ondelette 3D pour extraire les coefficients de première échelle :

$$U_1(x, y, z) = |f * \psi_{j_1,\theta_1}(x, y, z)|.$$

Étape 2 : Extraction des Coefficients de Haut Niveau Les coefficients U_1 sont transformés à leur tour pour capturer des informations secondaires :

$$U_2(x, y, z) = |U_1 * \psi_{j_2,\theta_2}(x, y, z)|.$$

Ce processus peut être répété sur plusieurs niveaux m, formant une cascade hiérarchique.

On observe également que ces applications d'ondelettes ont les mêmes caractéristiques qu'un CNN avec l'application de couches de convolutions ce qui rappelle que la décomposition en ondelettes est à la base de la Computer Vision à base de CNN.

Étape 3 : Agrégation Invariante À chaque niveau, on applique un opérateur non linéaire pour essayer de fabriquer des représentations invariantes (ce qui suit est un exemple d'opération que l'on peut faire):

$$S_m = \int |U_m| \, dx \, dy \, dz.$$

Ce sont ces S_m que l'on peut utiliser pour des downstreams tasks.
Une fois cela dit, il reste à tester cette idée.

9.2.4 Test de l'idée

Malheureusement, les contraintes de temps n'ont pas pu nous permettre d'explorer cette idée jusqu'au bout, en raison notamment de difficultés liées à l'optimisation de notre modèle via DINO sur le super-calculateur. Néanmoins, nous avons pu mener de petites expériences en utilisant les ondelettes de Haar en considérant une seule échelle de décomposition et en considérant l'"Approximation" de l'image.

Malgré ce manque de temps, l'entraînement a révélé un certain potentiel. Voir la figure 27:

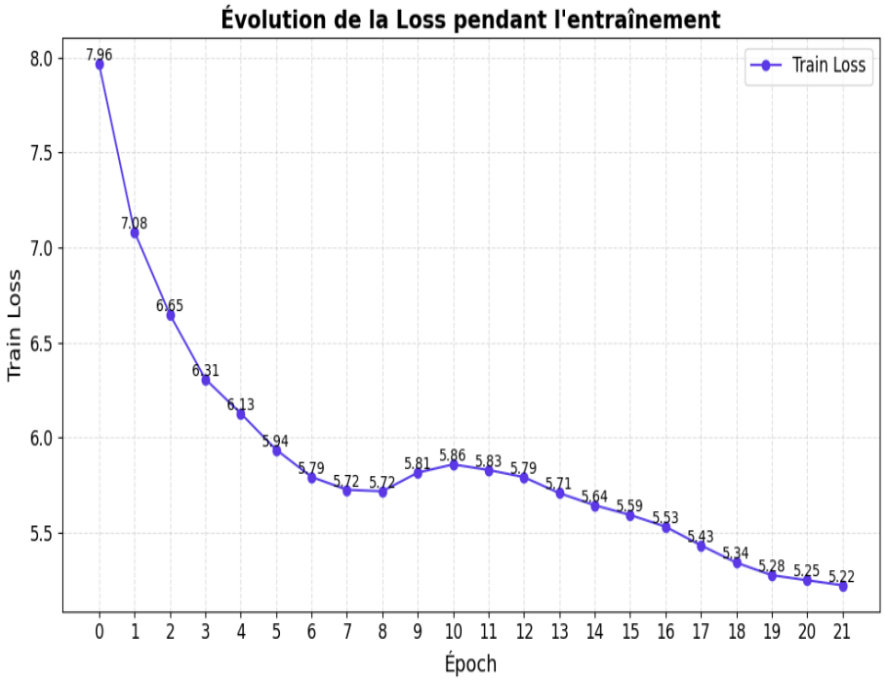

Figure 27: Fonction de coût d'entraînement avec la transformée en ondelettes

Malgré certaines diffcultés avec le supercomputer, l'idée de transformation en ondelettes semble prometteuse nottament vue la figure 27. Il faut maintenant tester sur les différents benchmarks si les résultats sont au rendez-vous. Ce travail pourrait être exploré par un prochain stagiaire.

10 Conclusion

En conclusion, on a pu, lors de cette étude, explorer une méthode innovante pour détecter des cancers dans des images d'Histopathologie de Prostate. On a ainsi pu comparer plusieurs façons de procéder en mettant en compétition des "foundation models" entraînés sur des grands datasets privés, contenant des images d'Histopathologie d'un grand nombre d'organes, avec des modèles entraînés uniquement sur des images d'Histologie de Prostate. On a ainsi pu voir que les résultats pour notre modèle entraîné sur le dataset PANDA (via MareNostrum 5, le super-calculateur du BSC) était en mesure de rivaliser avec ces foundation models qui eux n'avaient pas fait un focus sur l'organe de la Prostate.

On a également pu explorer des pistes d'améloriations pour notre propre modèle. Néanmoins, les contraintes de temps n'ont pas pu nous laisser le temps d'explorer en détails ces pistes d'amélioration. Bien que certaines expériences ont pu être menées.

References

[1] Joann G Elmore, Variability in Interpretive Performance at Screening Mammography and Radiologists' Characteristics Associated with Accuracy, 2009.

[2] Doersch, Unsupervised Visual Representation Learning by Context Prediction, ICCV, 2015.

[3] Devlin, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

[4] Chen, A Simple Framework for Contrastive Learning of Visual Representations, 2020.

[5] He, Momentum Contrast for Unsupervised Visual Representation Learning, 2019.

[6] Dosovitskiy, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021.

[7] Caron, Emerging Properties in Self-Supervised Vision Transformers, 2021.

[8] Jegou, DINOv2: Learning Robust Visual Features without Supervision, 2024.

[9] El Nouby, XCiT: Cross-Covariance Image Transformers, 2018.

[10] Pohjonen, HistoEncoder: a digital pathology foundation model for prostate cancer, 2024.

[11] Wenzek, CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data, 2020

[12] Zhou, Conditional Prompt Learning for Vision-Language Models, 2022